AD-762 444

A MICROPROGRAMMING LANGUAGE FOR THE MLP-900

Donald R. Oestreicher

University of Southern California

Prepared for:

Advanced Research Projects Agency

June 1973

(1)

1-B

ARPA ORDER NO. 2223/1

ISI/RR—73—8
June 1973

Donald R. Oestreicher

A MICROPROGRAMMING LANGUAGE FOR THE MLP-900

UB

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

10

Security Classification

DOCUMENT CONTROL DATA - R & D	
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)	
1 ORIGINATING ACTIVITY (Corporate author) USC Information Sciences Institute 4676 Admiralty Way Marina del Rey, California 90291	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED; 2b. GROUP -----
3 REPORT TITLE "A Microprogramming Language for the MLP-900"	
4 DESCRIPTIVE NOTES (Type of report and inclusive dates) Research Report	
5 AUTHOR(S) (First name, middle initial, last name) Donald R. Oestreicher	
6 REPORT DATE May 1973	7a. TOTAL NO. OF PAGES 9; 7b. NO. OF REFS 6
8a. CONTRACT OR GRANT NO DAHC15 72 C 0308; b. PROJECT NO ARPA Order #2223/1; c. Program code No. 3D30 & 3P10; d.	9a. ORIGINATOR'S REPORT NUMBER(S) ISI/RR-73-8; 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None
10 DISTRIBUTION STATEMENT Approved for release; distribution unlimited	
11 SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209

13 ABSTRACT

This paper describes a language for programming a microprocessor which combines the features of assembly languages with those of higher level languages. The goal of the language design was to provide a convenient microprogramming language for the MLP-900 microprocessor project at USC/Information Sciences Institute.

The goal was accomplished by designing a language with careful consideration of the hardware instruction set. The language was also constrained not to implicitly affect the machine at runtime. The considerations provided freedom and low-level control for the programmer. The flexibility needed by the compiler to allow for higher-level language forms was also provided by allowing the language to produce several microinstructions for each language statement.

****

14. KEYWORDS:

High-level language, implementation language, microprocessor, microprogramming, microprogramming language, MLP-900.

DD FORM 1 NOV 65 1473

Security Classification

ARPA ORDER NO. 2223/1

ISI/RR--73—8
June 1973

Donald R. Oestreicher

A MICROPROGRAMMING LANGUAGE FOR THE MLP-900

INFORMATION SCIENCES INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223/1, PROGRAM CODE NO. 3D30 AND 3P10.

Abstract

This paper describes a language, for programming a microprocessor, which combines the features of assembly languages with those of higher-level languages. The goal of the language design was to provide a convenient microprogramming language for the MLP-900 microprocessor project at the USC/Information Sciences Institute.

This goal was accomplished by designing a language with careful consideration of the hardware instruction set. Additionally, the language was constrained not to implicitly affect the machine state at runtime. These considerations provided freedom and low-level control for the programmer. The compiler needed some flexibility to allow for higher-level language forms. This flexibility was provided by allowing the language to produce several microinstructions for each language statement.

This project is sponsored by the Advanced Research Projects Agency. This work is directed toward an ARPANET-based sharable resource as a means of exploring computer architecture, language development and special purpose processor design, all of which are of particular relevance to DOD selection and use of computer equipment.

A MICROPROGRAMMING LANGUAGE FOR THE MLP-900

Donald R. Oestreicher

Introduction

Microprogrammed computers are typically characterized by small control memories for the storage of microprogrammed routines. These routines are used to implement the firmware instruction set for the target computer. The storage requirements and target computer instruction execution time considerations bring pressure on the microprogrammer to make optimal use of the microprocessor and associated control memory.

These conditions make microprogramming language designers and/or programmers tend towards a one-to-one correspondence between language statements and actual hardware microinstructions. As a result microprogramming languages often look like classical assembly languages [*1,2*]. This paper reports an effort to provide the convenience and readability of a higher-level language, without preempting the flexibility and machine state control available in assembly code [*3*].

General Purpose Microprogramming Language (GPM) is the primary language for the MLP-900 microprogramming project at the USC/Information Sciences Institute. The project's goal is to provide time-shared user access to a writable control memory microprocessor as a service in a multiprogrammed environment. This service is intended to be used in-house, as well as nationally over the ARPANET.

The MLP-900 is connected to a PDP-10 processor through the I/O buss (event channel) and the memory buss (data channel). The data bandwidth is 100MHz. The MLP-900 is strictly a slave processor. The PDP-10 TENEX time-sharing system does all target memory allocation and I/O for the MLP-900. The intention is for the MLP-900 to act as a user-specified time-shared execution engine for users on the PDP-10.

The MLP-900

GPM has been designed around the actual MLP-900 hardware. This has been done for efficiency and convenience. First, language forms ill-suited for the MLP-900 hardware were not provided, for efficiency. Second, special language forms were created to deal with the novel aspects of the MLP-900, for convenience. For this reason, a brief description of the MLP-900 is necessary for background to understand GPM.

The MLP-900 [4,5,6] is a vertical word microprocessor which runs synchronously with a 5MHz clock. It is characterized by two parallel computing engines called the Operating Engine (OE) and the Control Engine (CE). The OE performs arithmetic operations and the CE performs control operations. The OE contains 32 36-bit general-purpose registers (R0-R31) for operands and 16 36-bit mask registers (M0-M15) to specify operand fields. The CE contains 256 state flip/flops (F0-F255) organized in 16 8-bit registers (CE0-CE15). The CE also contains a 16 word hardware stack and 16 8-bit pointer registers (P0-P15). The writable control memory contains 4K words. Additionally, there is a 1K 36-bit auxillary memory.

The OE and CE will execute in parallel if, and only if, a CE instruction follows an OE instruction during the execution. Programmer consideration of this feature usually is not requireo However, if a program is executed entirely as OE-CE instruction pairs, the effective machine speed is doubled.

The MLP-900 also has features to support a microvisor, which will swap users and handle I/O requests to TENEX. These features include microvisor mode, privileged instructions, control memory protection, and processor state protection. Additionally, the MLP-900 has an address transformation box to allow demand paging of the target program and data in cooperation with the TENEX time-sharing system.

GPM Goals

The primary goal of the GPM design was to produce a higher-level language which did not preclude any coding options. In particular, as GPM was to be the primary language for the MLP-900, every control memory code had to be possible as language output. The language had to be amenable to diagnositc programmers, application programmers, and researchers. This was accomplished by combining the appropriate features of assembly code with the complimentary higher-level language features.

Some GPM statements look very much like assembly language. These statements correspond to the I/O instruction. The higher-level statements fall into four categories of interest:

1. syntactic block structure;
2. hardware generalization;
3. multi-instruction statements;
4. expressions.

Each of these will be discussed in detail below.

Syntactic Block Structure

The low-level constraints in the GPM design precluded the implementation of any dynamic storage allocation, or even of an operand stack. As a result, the block structuring in GPM may be considered to be a compile-time artifact. None the less, by using the block structure syntax in a most rudimentary way, the resulting language has been rendered more tractable and comprehensible.

The block structure syntax is the standard BEGIN declarations; body; END. This syntax is used at compile-time to specify scope. This is used both for data names (all labels are global) and control statements. Blocks may be named, and several blocks may be closed with a single named END.

Names

Every memory cell in the MLP-900 is explicitly named with a reserved word in GPM (e.g. general register 3 is R3 and pointer register 5 is P5). These names are not necessarily mnemonic, so the user may rename any memory cell at the top of a block.

Any synonym defined at the start of a block is undefined at the end of the block. This allows procedures to give mnemonic names to parameters and temporary registers. Additionally, the practice of "declaring" registers at the top of a block renders the blocks scope instantly apparent.

This block-structured synonym facility, if exploited properly, can produce more readable programs. The user may also rename any reserved word in GPM using this same facility.

Control Statements

The block structure syntax is used to define the scope of IF statements and DO statements. The DO statement heads a block which will be iterated upon indefinitely. The method of exiting a DO block is the BREAK statement.

The BREAK statement semantics are defined in terms of the lexical block structure. A BREAK statement transfers control out of the lexical block named by the statement. If no name is supplied, the current block is assumed.

The above examples of applications of block structure syntax demonstrate how the concept can be useful in a semantically simple language. One could even restate the GPM design goal: design a language which is syntactically rich and semantically poor.

Hardware Generalization

One of the more classical functions of a programming language is to provide a complete set of functions, where the hardware may not. For instance, on a computer with only a jump on less than zero, the programming language would provide all eight possible jumps relative to zero.

GPM attempts to do this in all cases where it is possible, without violating the design constraints. Two examples will illustrate this idea.

Example - GOTO destinations

The following are all hardware MLP-900 instructions:

GOTO 100;	absolute jump
GOTO + 10;	relative jump
GOTO 5 (P0);	indexed absolute
GOTO + 1 (P0);	indexed relative

However, the statement GOTO + 3 <P0> is not an MLP-900 instruction, as the hardware only supports indexed relative jumps with a + 1 relative offset. However, the above statement is legal in GPM and the compiler changes the relative offset to the appropriate absolute address.

Example - CE assignments

This example will require a further description of the vagaries of the MLP-900. As mentioned above, the CE contains 256 state flip/flops (F0-F255) organized into 16 8-bit registers (CE0-CE15). This example discusses the instructions to transfer data between these entities. The following are all hardware MLP-900 instructions.

```
CE0 ← CE1 (77) ;
```

Transfer the contents of register 1 to register 0 for all bits set in the octal mask (77 in this case).

```
CE0 ←  NOT CE1 (77) ;
```

This is the same as the previous instruction, except register 1 is complemented first.

```
CE0 ← CE1 [77] ;
```

This is the same as the first instruction, except all bits not set in the mask are cleared in register 0.

The legal GPM statement

```
CE0 ← NOT CE1 [77] ;
```

will compile into*

```
CE0 ← NOT CE1 (77) ;
CE0 ← CE0 [77] ;
```

This type of language feature allows the programmer to ignore some of the intricacies of the hardware. However, if a GPM programmer wishes, he/she may stick to the GPM subset which corresponds to the hardware MLP-900 instructions.

* As GPM is the primary MLP-900 language, it compiles into the subset of itself which corresponds to actual hardware instructions

Multi-instruction Statements

Some GPM statements will always compile into several hardware MLP-900 instructions. These are common fuctions with which the user should not have to concern himself/herself.

Example - Case statement

The GPM statement

```
SWITCHON  P0 INTO
```

will produce an indexed jump into a transfer table specified by CASEs specified in the block which the SWITCHON heads. This requires the automatic generation of the transfer table somewhere in control memory.

Example - Inter-engine assignments

In order to transfer data from the operating engine to the control engine, the exchange buss (XBUS) is used. This requires an OE-CE instruction pair to be executed in parallel. Therefore, all inter-engine assignments require the generation of two instructions.

The GPM Statement

```
CE0 ← R0 ;
```

will compile into

```
XBUS ← R0 ;
CE0 ← XBUS ;
```

However, as both instructions are to be executed in parallel, the GPM statement

```
R0 ← CE0 ;
```

will compile into the non-intuitive

```
R0 ← XBUS ;
XBUS ← P0 ;
```

These multi-instruction generating statements make programs shorter and thus easier to read. The information not explicitly stated is essentially irrelevant in the latter example, and redundant in the former. Thus, brevity is achieved without the introduction of obscurity.

Expressions

Expressions are so common in higher-level languages, it might seem out of place to devote an entire section to them here. However, the constraints on the GPM design complicates the compilation of expressions. GPM is not allowed to make any implicit changes to the machine state at runtime. This precludes the introduction of temporaries to evaluate expressions. Two brief examples will demonstrate how expressions are to be handled.

Example - Arithmetic expressions

The GPM statement

```
R0 ← R0 AND R1 + R2 ;
```

will compile into

```
R0 ← R0 AND R1 ;
R0 ← R0 + R2 ;
```

However the GPM statement

```
R0 ← R0 AND ( R1 + R2 ) ;
```

will not compile, for lack of a temporary for (RI + R2).

Example - Boolean expressions

The GPM statement

```
F0 ← (F1 and F2) or ( F3 AND F4) ;
```

will compile into

```
IF  F1 AND F2  THEN GOTO +3 ;
IF  F3 AND F4  THEN GOTO +2 ;
IF  NOT (F0 ← FALSE )  THEN
GOTO +2 ;
F0 ← TRUE ;
```

Boolean expressions will always compile as the program counter can be used as a temporary boolean value.

The expression evaluation in GPM leaves much to be desired, but it was felt that when the expressions worked, they were so superior to the assembly code alternative that they would be included in spite of themselves.

Conclusion

This paper has reported on some ideas to make microprogramming more agreeable in light of the previous experience of the computer community with conventional computers. The opinion stated here is that a hybrid language is necessary for the task. This paper described several of the problems and solutions associated with this approach.

It appears that with careful consideration to the actual processor in question, it is possible to create a passive higher-level language, which allows total user control, while, at the same time, encouraging readable programs and allowing easy language usage.

References

1. Dubbs, E. W., Parsons, R. L., Peterson, J. E., "A Microprogram Design System Translator," in Sixth Annual IEEE Computer Society International Conference, *Digest of Papers*, San Francisco, California, September 12-14, 1972, pp. 95-98.

2. Berndt, Helmet, "Microprogramming with Statements of Higher-Level Languages," in *Fifth Annual Workshop in Microprogramming*, Urbana, Illinois, September 25-26, 1972, pp. 76-80.

3. Eckhouse, R. H., "A High-Level Microprogramming Langua[illegible]," in *Proceedings of the Spring Joint Computer Conference, 1971*, AFIPS Proc., Vol. 38, Montvale, New Jersey: AFIPS Press, 1971.

4. STANDARD Computer Corporation, *MLP-900 Multi-Lingual Processor - Principles of Operation*, Santa Ana, California: STANDARD Computer Corporation, Technical Publications, 1970.

5. Lawson Jr., H. W., Smith B. K., "Functional Characteristics of a Multilingual Processor," *IEEE Transactions on Computers*, Vol. C-20, No. 7, July 1971, pp. 732-742.

6. Guffin, Ronald M., "Microdiagnostics for the STANDARD Computer MLP-900 Processor," *IEEE Transactions on Computers*, Vol. C-20, No. 7, July 1971, pp. 803-808.